Software component reuse in information systems development: a review of challenges and strategies

Gerald Goh Guan Gan
School of Multimedia & IT
Han Chiang College
11600 Penang MALAYSIA

Wong Kim Yen
School of Multimedia & IT
Han Chiang College
11600 Penang MALAYSIA

Assoc Prof Mark Toleman
Department of Information Systems
Faculty of Business
University of Southern Queensland
QLD 4350 AUSTRALIA

The development of information systems projects has always been plagued by high incidences of failure which can be attributed to the sheer complexity of the problem at hand coupled with uncertainties brought about by the dynamic business environment and constantly evolving technologies. With component-based software development (CBSD), organisations can now utilise component technology to increase the likelihood of project success. This is made possible by component reuse which can be leveraged to minimise the effort required to develop applications. Component reuse needs to be organised and implemented well by the adopting organisation to ensure success. To maximise the likelihood of success, the organisation must ensure that the effort is systematically planned and addresses a wide array of technical as well as non-technical issues. More importantly, the organisation must not be fixated on the technology itself; instead adequate attention must be given to the non-technical issues which are often sidelined by project managers. This paper will briefly introduce the key concepts of CBSD and component reuse before providing a detailed discussion on the various technical and non-technical issues that need to be considered in software component reuse. The main contribution of this paper is that it provides a discussion of the issues that need to be considered by the organisation in implementing component reuse programmes.

Keywords: *Software reuse; component-based software development; IS project management*

Goh, GGG, Wong, KY & Toleman, M 2005, 'Software component reuse in information systems development: a review of challenges and strategies', *Journal of Han Chiang College*, vol. 3, pp. 83 – 95.

1.0 Introduction

Due to the complexity, limited resources and at times chaos that exist in most information systems development efforts, incidences of project failure are high illustrating the crisis faced by the software industry these days (Ewusi-Mensah 1997; Jones 1995; Martin & Chan 1996; Tiernan 2003). According to McBride (2004), up to one-third of all money spent on information systems development is used to repair botched projects with billions spent each year reworking software that does not fit customer requirements.

A landmark survey on information systems project failure by the Standish Group (1995) which canvassed the views of IT executive managers from large, medium and small companies across major industry segments - banking, securities, manufacturing, retail, wholesale, heath care, insurance, services, and government organizations reveal that a huge amount of losses is incurred due to these failures. The Standish Group research shows a staggering 31.1% of projects will be cancelled before they ever get completed. The cost of these failures and overruns are immense as the lost opportunity costs are not immediately quantifiable, but could easily be in the trillions of dollars in the United States alone (The Standish Group 1995, pp. 1-2).

Upon examination it is found that the root cause is attributable to poor project estimating techniques, ineffective project team structure, a lack of client project participation, unmanaged scopes and taking on large-scale project risk without any contingencies (Tiernan 2003; McBride 2004). In addition to that, 53% of projects completed are 188% over budget at an additional cost of USD$ 59 billion with larger projects completed with only 42% of their original specifications or user requirements (Misciagna 2002).

On the success side, the Standish Group (1995, p. 2) found that the average is only 16.2% for software projects that are completed on-time and on-budget. In the larger companies, the news is even worse with only 9% of their projects come in on-time and within-budget. Even when these projects are completed, many are no more than a mere shadow of their original specification requirements (The Standish Group 1995, p. 2). It was found that projects completed by the largest American companies have only approximately 42% of the originally-proposed features and functions (The Standish Group 1995, p. 2). It is undeniable that information systems project failure is devastating to an organization. Schedule slips, 'buggy' releases and missing features can mean the end of the project or even financial ruin for a company (McBride 2004; Misciagna 2002; The Standish Group 1995). As such, measures to overcome this perennial problem of failure need to be addressed by industry practitioners.

To effectively ensure systems development success, component-based software development has been hailed by many software developers and academics as being one of the most promising ways of controlling the increasing costs and complexity of most information systems (Herzum & Sims 2000). In addition, the economic and technological realities of the 1990s have led to many changes to the enterprise, especially in terms of the need for software to be flexible in accepting new technology upgrades, the growth in distributed computing and the downsizing, restructuring and re-scoping of organisations (Herzum & Sims 2000; Brown 1996).

In view of these developments, Brown (1996) posits that it would be infeasible for developers and organisations to consider constructing each new information system from scratch. Instead, information systems would need to be developed with reused practices, software components and products that have been tested and proven to be effective and efficient in order to remain in business and gain competitive advantage (Brown 1996; Butt 2001; Szyperski 1998). There are two key elements to designing for reuse – designing software components that can be reused and

reusing components that already exist. Due to the importance of reusing software components, it is therefore crucial to understand the issues and challenges faced when developers reuse software components to enable effective reuse programmes to be incorporated into the information systems development effort in the enterprise.

This paper first provides a definition of software components and outlines its major features with regard to information systems development. Next, the importance and potential gains of reusability afforded by component software in information systems development are identified and appraised. Having understood the benefits of reuse in component-based systems development, this paper will then discuss some insightful strategies that can be adopted by information systems developers to ensure successful reuse of software components in their projects to achieve improved system quality in addition to timely delivery of complex systems.

2.0 What is a software component?

According to Herzum and Sims (2000), the term 'component' is used in many different ways by practitioners in the industry. However, a rather broad and general yet useful definition proposed by Brown (1996) defines a software component as 'an independently deliverable piece of functionality providing access to its services through interfaces'. In addition, it is important for a software component to be easily combined and composed with other software components (Herzum & Sims 2000). This is because a software component will only achieve its usefulness when it is used in collaboration with other software components (Herzum & Sims 2000; Szyperski 1998). The interface is how a user of the component views the component. The user is concerned with the interface that they are using and since the component is the implementation of an interface, the user is actually concerned with how the interface behaves.

Among the characteristics of a software component that facilitates collaboration is that it is a self-contained software construct that has a defined use, has a run-time interface, can be autonomously deployed, and is built with foreknowledge of a specific component socket (Herzum & Sims 2000). Herzum and Sims (2000) add that components are built for composition with other components and have well-defined and well known run time interface to a supporting infrastructure. A design-time interface alone is necessary but not sufficient because it does not exist in the run-time unless it is implemented by some piece of software, that is, by infrastructure (Herzum & Sims 2000).

Apperly (1999) asserts that the use of software components or component-based software development allows organisations to develop innovative information systems solutions within the stipulated time and budget. Both quality and time-to-market the solution is greatly enhanced by using software components that were pre-built and effectively reused by the development team (Apperly 1999; Griss 2001; McInnis 2002; O'Donnell 2001).

3.0 Reusing software components

Reusable software components are not new practice but are becoming more widespread due to emergence of component-based technologies such as Microsoft COM+, Enterprise Java Beans and the CORBA Component Model (Brown 2000). Many organisations now practise software reuse by assembling pre-existing components (within or across domains) when developing new components or information systems (Sitaraman et al. 2002). Sitaraman, Long, Weide, Harner and Wang (2002) contend that component reuse is a basic tenet and a key feature of component-based development.

Component reuse refers to the process of implementing or updating software systems using previously created or existing assets (Lim 2002; Pinto 2002). Williams (2001) stresses that software reuse is something that has gained widespread attention of software developers for years but has failed to be fully practised to a significant degree. Fortunately, component-based software development strongly supports reuse and this effectively paves the way for the benefits of reuse to be accrued by organisations now (McInnis 2002; O'Donnell 2001; Williams 2001). As such, the benefits of reusing software components in component-based development are detailed in the following section.

3.1 Benefits of software reuse

As mentioned in the preceding section, there are many benefits of reusing software components in information systems development. When correctly applied and implemented, reuse can increase productivity, shorten time-to-market, improve software quality, reduce maintenance cost, allow for inter-application interoperability, reduce risks, leverage technical skills and knowledge, and improve system functionality (Lim 2002; Patrizio 2000; Pinto 2002; Schmidt 2002).

Developing software using traditional software development approaches will more often than not result in organisations facing application backlogs. This is because changing one part or even one statement of software code will have adverse effects to other parts of software. Hence, software developers have limited time to develop new systems because they have to expend their existing resources to maintain existing systems (Pressman 1997). Due to its inherent power of functional independence in CBSD where software is assembled from self-contained components that can be autonomously deployed, the productivity and performance of the development team can be improved further (Herzum & Sims 2000).

According to Lim (2002), Hewlett-Packard software projects reported productivity increases from 6% to 40% with the incorporation of CBSD. Pitney Bowes in the USA which has been reusing components since 1996 documented tremendous savings in labour as the company is now able to achieve 500 human-weeks of development progress in only 200 human-weeks by using existing components and by purchasing others from component markets (Scannell 2002).

Apart from productivity gains, component reuse allows organisations to reduce the critical path in the delivery information systems applications, reducing the time-to-market and begin to accrue profits earlier (Lim 2002). With proper planning of component interfaces and infrastructure design, different development teams at different locations can develop their own components simultaneously (Herzum & Sims 2000; Brown 2000). Besides that, software systems can assemble components across boundaries at run time or design time which encourages distributed software development (Herzum & Sims 2000).

The quality of information systems developed using this approach will also have fewer bugs and defects if compared with newly built-from-scratch systems (Pinto 2002). From a cost perspective, if an asset's costs can be amortised through a large number of uses, it would then be possible for the management to expend more effort and allocate more budget to improve the quality of software components (Pinto 2002). This in turn reduces the level of risk faced by the development effort and will undeniably improve the likelihood of success (Lim 2002).

Maintaining legacy systems is a nightmare for every organisation (Pressman 1997). Almost 80% of software development costs are used to maintain the systems after they have been implemented (O'Donnell 2001; Pressman 1997).Therefore, one major advantage of a component is its plug and play feature which allows easy composition and inclusion in the information systems effort (Brown

2000; O'Donnell 2001). Organisations can throw away unwanted components and assemble them with more advanced components based on their needs without affecting the functions of other components (Szyperski 1998).

In addition, when systems are developed using reused components, they are expected to be more interoperable as they rely on common mechanisms to implement most of their functions (Pinto 2002). Dialogs and interfaces used by these systems would be similar and would improve the learning curve of users who utilise several different systems built using the same components (Lim 2002; Pinto 2002). Software reuse in CBSD also allows specialists to optimise the software components and the component-based development architecture being developed which could then be reused by other developers whose main tasks would be meeting the product feature needs and the required functionality as specified by the users (Lim 2002).

Due to the impressive potential benefits of reuse, the Gartner Group anticipated that almost 70% of new information systems by 2003 will be assembled using reused and pre-written software components (Vinas 2002, p. 67). Hence, it is exigent that organisations implement the correct strategies to ensure the adoption and continued use of reused software components in a systematic manner.

4.0 Strategies for implementing reuse in component-based software development

It is claimed that software reuse advocates and practitioners have learned that careful attention needs to be paid to both technical and also non-technical issues in order to ensure success in software component reuse (Griss 2001; PC Week 1997). In most instances, non-technical issues proved to be more pervasive and complex than realised by many people (Griss 2001). The top nine obstacles or problems (see Table 1) often faced when implementing software reuse adoption in organisations can be categorised into technical (structure mismatch, steep learning curves), managerial problems (infrastructure clash, turf battles, inadequate resources) and cultural or psychological issues (apathy, not invented here syndrome, fear and ivory towerism) (Reifer 2001).

Table 1 Barriers to software component reuse

Technical	Managerial	Cultural/Psychological
• Structure mismatch • Steep learning curves	• Infrastructure clash • Turf battles • Never enough time and money (inadequate resources)	• Apathy • Not invented here syndrome • Fear of the unknown • Ivory towerism

(Source: Griss 2001, p. 454)

The issue of structure mismatch emerges as most software development organisations have embraced paradigms, methods and tools that do not promote the use of software components (Griss 2001). These ingrained elements result in these organisations attempting to build systems from scratch and not reusing available components (Griss 2001). CBD is further complicated by the lack of tools needed to build systems using templates, software development models, software component infrastructures and building block libraries. As such, guidelines need to be developed and CDB tools acquired to encourage the use of CBD in organisations (Griss 2001).

Due to the conceptual mismatches of CBD and current development practices, a steep learning curve for component reuse exists (Griss 2001). The minds of developers need to be 'open' to ensure that they are able to embrace new ways of doing business and experience indicates that it may take up to 12 – 18 months to become proficient and competent in CBD (Lim 1998). To overcome this, it is advisable to integrate both training and mentoring of CBD on smaller and less mission critical information systems projects (Griss 2001).

From a managerial perspective, most large and defence-related organisations have already established software management infrastructure consisting of Capability Maturity Model (CMM)-compatible processes and criteria upon which its organisational structures, operational concepts and reward culture based on (Griss 2001). This existing infrastructure needs to be altered to facilitate the introduction of CBD. This in turn would lead to major distractions as it may impact the manner in which decisions are made and work is done (Griss 2001).

When CBD is adopted, turf battles tend to ensue as responsibilities, budget allocations, power and the like are assigned (Griss 2001). As organisations compete for these resources, animosities may be created with acrimony lingering even after the turf battle appears to have been settled (Griss 2001). Consequently, adequate care must be taken to ensure that there is consensus over who does what in order to ensure that all levels of the organisation 'buys' into the concept of CBD (Griss 2001). Another barrier closely aligned to turf battles is the problem of inadequate resources. More often than not, there is inadequate time, talent, people and money to do things right (Griss 2001). Hence, proper allocation of resources is crucial to ensure that the task of adopting CBD is not hampered by spreading the organisation thin (Griss 2001).

The most common question posed to change agents and proponents of CBD is 'Why change when nothing is broken?' (Griss 2001). Apathy with regard to the adoption of new methods or tools is not a new issue and CBD proponents can counter this response by building a case for component reuse using both business and technical perspectives (Griss 2001). A well-planned business plan can help change agents convince critics that it makes sense to change and that CBSE is in everyone's best interests (Griss 2001).

The not-invented here syndrome is another barrier that may be encountered as many individuals are of the opinion that solutions that work well in different organisations may not necessarily work well in their organisation (Griss 2001). One way of overcoming this is to cite success stories and to bring critics on a visit to view successful CBD implementations that they can relate to easily (Griss 2001). In addition, people are often fearful of the unknown. They must be persuaded to embrace change and be able to appreciate the potential benefits of this initiative (Griss 2001).

Finally, ivory towerism may have its impacts felt when trying to encourage CBD adoption as many developers may be of the view that CBD is an immature technology with very little to benchmark and that it may drain resources from other projects (Griss 2001). CBD proponents would therefore need to be able to demonstrate that this is untrue by getting successful organisations in the CBD approach to share their experiences and thought on using CBD (Griss 2001). In view of these barriers identified by Griss (2001), it is crucial that strategies be formulated to address these barriers and to ensure successful deployment of CBD technology in organisations.

One of the more important strategies for implementing reuse is the cultivation of an appropriate 'reuse mindset' proposed by Griss (2001). A correct mindset and attitude must be in place to ensure that everyone who is involved in the development of components understands the motivations and potential benefits of component-reuse to the organisation. This will in effect eliminate the common

'not invented here syndrome' problem where developers have the misconception that reused software components from other projects will not work on their current project as it was invented somewhere else (Reifer 2001).

To succeed, developers must create a groundswell for technological innovation, encourage project managers and team members to do things differently. The support of the top management and also the other levels of the organisation involved in the reuse effort is a precondition for success (Lim 2001; Reifer 2001). Schmidt (2002) observed that the ability of component reuse to succeed is highly correlated with the quality and quantity of effective leaders and experienced developers. In general, highly experienced and skilled developers who are empowered to create, document and support horizontal middleware platforms are required in order to allow high-1evel application developers who need not be as experienced as complex systems-level technologists to focus on programming higher-level abstractions with reused components (Schmidt 2002).

Comment [c1]: technology?

Another area that needs adequate attention is the corporate culture which must be supportive of reuse initiatives and efforts (Schmidt 2002; Sitaraman et al. 2002). Creating incentives allows developers to share in the benefits that they have contributed and provides a more immediate return on the organisation's component reuse investment (Poulin 2002).

The incentives awarded by the organisation should ultimately be tied to reuse metrics to bridge the gap between project-level metrics and the developers' daily work (O'Donnell 2001; Poulin 2002). This would allow them to observe how their reuse actions make a significant impact on the well-being of the organisation and allow them to gain some recognition for their effort. Among the incentives that can be used are public recognition, presentation of tokens or gifts and perhaps even participation in reuse conferences or holiday packages to the value that the developer has brought to the organisation (Pinto 2002; Poulin 2002). In short, these incentives act as 'carrots' to motivate developers and to generate interest in component reuse within the organisation.

Apart from the soft issues covered, a variety of technical factors must also be addressed. This includes quality assurance (QA) of components being developed, proper documentation standards and the use of appropriate component-based frameworks. Adhikari (2002) stresses that quality assurance cannot be an afterthought and needs to be built-in from the beginning to ensure that the components to be reused are of sound quality and have been subjected to rigorous testing to isolate errors.

CBSD is not like traditional software development techniques in that it focuses on interface reuse instead of code reuse (Due 2000). To facilitate reuse, proper and comprehensive documentation regarding the implementation interface and the functionality of components must be in place (Houston & Norris 2001). In addition, well-specified contracts (preconditions and postconditions to using interfaces and components) of the required interface with the contracts of the existing components are essential for CBSD (Due 2000). With this information properly documented, it is easier for developers to locate and reuse the appropriate software components (Houston & Norris 2001).

Clear and unambiguous requirements definition is a precursor to CBSD success. Requirements specification can be used to determine the quality of software component design (Pressman 1997). Therefore, software component design should be walked-through based on each scenario to determine how well the design meets with stipulated requirements (Bass 2001). If the designer is able to check the completeness and correctness of a design based on its system requirements at an

early stage, then it will reduce the likelihood of problems encountered in the later stages and consequently reduce the costs, time and effort to correct the problems (Stafford & Wolf 2001).

In CBSD, every component should achieve functional independence where providing access to its services through interfaces (Brown 1996). Herzum and Sims (2000) provide a similar concept where 'emphasis of component-based design is autonomy, which means components should perform complete functions through minimal interfaces needed to connect the components into a larger information system'. Besides, if it is clear that a component is performing more than one task then the component should be split into several components to make sure each component performs a single unique function Pressman 1997; Scannell 2002).

By promoting functional independence, a component has greater potential for reuse and to facilitate independent software development (Sitaraman et al. 2002). As different project teams are required to develop large-scale distributed systems, functional independence of a software component is necessary for easier assembly with other components by the way to decouple developers and users of components (Herzum & Sims 2000; Sitaraman et al. 2002). Besides, designers can design reusable components by looking for common behaviours that exist in more than one place in the system and generalising their function for reusability (McInnis 2002).

Due to advancement in network technology and distributed computing, future information systems applications will more frequently be constructed by assembling components across a range of machines through the network rather than being limited in a local machine (Brown 2000). Hence, scalability of components for distributed systems must be taken into consideration (Herzum & Sims 2000). The important factors that need to be considered are 'how information is communicated between components and how components are referenced, and to minimise the number of invocations across network' (Herzum & Sims 2000).

Nowadays, software architecting is an important activity to succeed in IS development. Software architecture can be defined as 'a level of design that specifies the overall system structure of a software application' (Shaw & Garlan 2002). In CBSD, software architecture encompasses the original component infrastructure as well as the non-CBS infrastructure to form a complete system (Councill & Heineman 2001). Bass (2001) suggests eight principles of architectural design to succeed in designing of solid software (refer to Table 2).

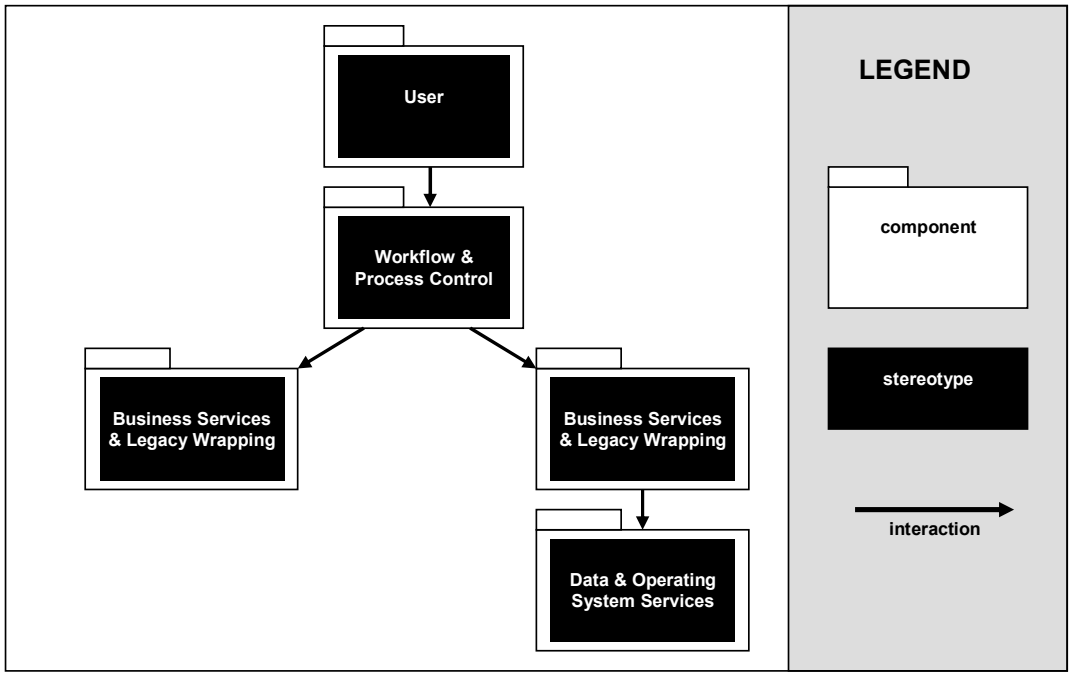
Table 2 Eight principles of architectural design

Explicitly identify both high level and concrete functional and quality requirements
Explicitly identify architectural drivers
Choose an architectural style that best satisfies architectural drivers
Divide functionality in a manner that supports quality requirements
Use concurrency and deployment views to identify functionality not previously considered
Verify using concrete quality and functional requirements
Identify the appropriate component model
Iterate to refine the design elements

(Source: Bass 2001, p. 401)

Most software components developed during the past few years have not been reused; one of obvious reasons is the poor design of infrastructure (Adhikari 2002). Therefore, it is necessary to design a set of component infrastructures that meets or exceeds its requirement or performance objective. Latchem (2001) suggests that 'the component infrastructure must be carefully designed to separate responsibilities and ensure that the logical connections between components do not result in unnecessary coupling'. To achieve this goal, a designer can establish a series of layers based on the various types of components with predefined responsibilities as illustrated in Figure 1

Figure 1 The component structure and their interconnections



(Source: Latchem 2001, p. 265).

Then, the designer can specify the component's stereotypes for each layer and describe their behaviour (Latchem 2001). The components can be categorised based on the services they perform or based on models of the business (domain-specific) within which components work (Latchem 2001; Wills 2001).

5.0 Conclusion

The development of information systems projects have always been plagued by high incidences of failure which can be attributed to the sheer complexity of the problem at hand coupled with uncertainties brought about by the dynamic economic environment and constantly evolving technology. With the introduction of component-based software development, organisations can now utilise component technology to increase the likelihood of project success. This is made possible by a key feature of component technology which is known as component reuse which needs to be organised and implemented well to ensure success.

In order to maximise the likelihood of success for component reuse programs, organisations must ensure that the effort is systematically planned and addresses the wide array of technical and non-

technical issues that abound. More importantly, the organisation must not be fixated on the technology itself; instead adequate attention must be given to the non-technical issues which are often sidelined by project managers.

References

Adhikari, R. 2002, 'The quest for component reuse carries on', *Application Development Trends*, February.

Apperly. H. 1999, 'CBD Techniques get business up to speed', *Computer Dealer News*, October, p 29.

Bass, L. 2001, 'Software Architecture Design Principles', in *Component-Based Software Engineering: Putting the Pieces Together*, eds G. T. Heineman & W. T. Councill, USA, pp. 389-403.

Brown, A. W, 1996, 'Preface: Foundations for component-based software engineering', in *Component-based Software Engineering - Selected Papers from the Software Engineering Institute*, ed A. W. Brown, pp. vii - x.

_____ 2000, *Large-Scale, Component-Based Development*, Prentice-Hall, USA.

Butt, J. 2001 , 'Reuse Rather Than Rebuild', *eWeek*, vol 1 8, no. 44, p. 37.

Councill. B. & Heineman, G. T. 2001, 'Definition of a Software Component and its Elements', in *Component-based Software Engineering - Putting the Pieces Together*, eds B Councill & G. T. Heineman, Addison Wesley, USA.

Dué, R. T. 2000, 'The Economics of Component-Based Development', *Information Systems Management*, winter, pp. 92-5.

Ewusi-Mensah, K. 1997, 'Critical issues in abandoned information systems development projects', *Communication of the ACM*, vol. 40, n0.9, pp. 74-80.

Graham, T. C. N., Morton, C. A. & Urnes, T. 1996, 'ClockWorks: Visual Programming of Component-Based Software Architectures', *Journal of Visual Language and Computing*, vol. 7, pp. 175-96.

Griss, M. L 2001, 'Product-Line Architectures', in *Component-based Software Engineering - Putting the Pieces Together*, eds B. Councill & G. T. Heineman, Addison Wesley, USA.

Herzum, P. & Sims, O. 2000, *Business Component Factory- A Comprehensive Overview of Component-based Development for Enterprise,* John Wiley, New York.

Houston. K. & Norris, D. 2001, 'Software Components and the UML', in *Component-based Software Engineering - Putting the Pieces Together*, eds B. Councill & G. T. Heineman. Addison Wesley. USA.

Jones, C. 1995, 'Patterns of Large Software Systems: Failure and Success', *Computer*, March, pp. 86-7.

Latchem, S. 2001, 'Component Infrastructures: Placing Software Components in Context' in *Component-Based Software Engineering: Putting the Pieces Together*, eds G. T. Heineman & W. T. Councill, USA, pp. 263-83.

Lim, W. C. 2001, *What is Software Reuse?* [Online], Available:http://www.flashline.com/content/lim/what_reuse.jsp, [Accessed : 20 August 2002].

Martin, A. & Chan, M 1996, ' Information systems project redefinition in New Zealand: Will we ever learn?', *Australian Computer Journal*, vol. 28, no. 1, pp. 27-35.

McBride, S 2004, 'Poor project management leads to high failure rate', Computerworld Today, 15 October, Australia.

McInnis, K. 1999, *Component-based Design and Reuse* [Online], Available: http://www.cbd-hq.com/articles/1999/990715_cbdandreuse.asp, [Accessed: 15 August 2002].

Misciagna, M. 2002, *Ensuring Project Success* [Online], Available:http://www.rdacustomsoftware.com/services/thought/marty.htm, [Accessed: 5 August 2002].

O'Donnell, A. 2001, 'Building Blocks of Success', *Insurance & Technology*, September, pp. 41-5.

Patrizio, A. 2000, 'The New Developer Portals - Buying, selling, and building components on the web speeds companies' time to market', *Information Week*, August, p. 81.

PC Week 1995, USA, p.68 in Ewusi-Mensah, K. 1997, 'Critical issues in abandoned information systems development projects', *Communications of the ACM*, vol. 40, no. 9, pp. 74-80.

Pinto, P. E. 1998, *Promoting Software Reuse in a Computer Setting* [Online], Available: http://www-2.cs.cmu.edu/afs/cs/usr/ppinto/www/reuse.html, [Accessed: 20 August 2002].

Poulin, J. S. 2001, *Creating Incentives for Reuse in Your Organisation* [Online], Available: http://www.flashline.com/content/poulin/creating_incentives.jsp, [Accessed: 22 August 2002].

Pressman, R. S. 1997, *Software Engineering: A Practitioner's Approach*, 4th edn., McGraw-Hill, USA.

Reifer, D. J. 2001, 'Implementing a Practical Reuse Program for Software Components', in *Component-based Software Engineering - Putting the Pieces Together*, eds B. Councill & G. T. Heineman, Addison Wesley, USA.

Scannell, E. 2001, *Web Services Reignite Component Reuse* [Online], Available: http://www.itworld.com/AppDev/4162/IWD010409hnreuse/pfindex.html, [Accessed: 15 August 2002].

Schmidt, D. C. 2001, Why *Software Reuse has Failed and How to Make it Work for You* [Online], Available: http://www.flashline.com/content/DCSchmidt/lesson_1.jsp, [Accessed: 18 August 2002].

Shaw, M. and Garlan, D. 1996, Software Architecture: Perspectives on an Emerging Discipline, quoted in Brown A. W. 2002, *Component-Based Software Engineering – Selected Papers from the Software Engineering Institute*, USA, p. 368.

Sitaraman, M. Long. T. J., Weide, B. W., Harner, E. J. & Wang, L. 2002, 'Teaching Component-based Software Engineering: A formal approach and its evaluation', *Computer Science Education*, vol. 12, no. 1-2, pp. 11-36.

Stafford, J. A. & Wolf, A. L. 2001, 'Software Architecture', in *Component-Based Software Engineering: Putting the Pieces Together*, eds G. T. Heineman & W. T. Councill, USA, pp. 371-87.

Szyperski, C. 1998, *Component Software - Beyond Object-oriented Programming*, ACM Press/Addison Wesley, USA.

The Standish Group 1995, The Standish Group Report – Chaos [Online], Available: http://www.projectsmart.co.uk/docs/chaos_report.pdf, [Accessed: 28 December 2005].

Tiernan, C 2003, 'How should risk be factored into discount rates when assessing IT investments?', IMIS Journal, vol. 13, no. 2, pp. 25-6.

Vinas, T. 2002, 'Reduce through reuse', *Industry Week*, vol. 251, no. 2, p. 67.

Williams. J. 2001, 'The Business Case for Components', in *Component-based Software Engineering – Putting the Pieces Together*, eds B. Councill & G. T. Heineman, Addison Wesley, USA.

Wills, A. C. 2001, 'Components and Connectors: Catalysis Techniques for Designing Component Infrastructures', in *Component-Based Software Engineering: Putting the Pieces Together*, eds G. T. Heineman & W. T. Councill, USA, pp. 307-19.

Biographical Notes

Gerald Goh Guan Gan is the Head of the School of Multimedia & Information Technologies at Han Chiang College, Penang, Malaysia. He has a Diploma in Computer Studies (Distinction) and Advanced Diploma in Computer Studies (Distinction) from the National Computing Centre, UK and a Bachelor of Information Technology (Distinction) from the University of Southern Queensland, Australia. He has also completed a Master of Arts (Communication) from Universiti Sains Malaysia, Graduate Diploma in Information Systems and Master of Business Information Technology from the Australian Graduate School of Business, University of Southern Queensland. He is currently enrolled in a PhD program in knowledge management at Universiti Sains Malaysia. His research interests include electronic commerce, knowledge management, component-based software engineering, communications technology and media studies.

Wong Kim Yen is a Lecturer in the School of Multimedia & Information Technologies at Han Chiang College, Penang, Malaysia. He has a Diploma in Computer Studies and Advanced Diploma in Computer Studies from the National Computing Centre, UK and a Bachelor of Information Technology from the University of Southern Queensland, Australia. He has also completed a Master of Business Information Technology from the Australian Graduate School of Business, University of Southern Queensland. His research interests include electronic commerce, component-based software engineering, applications development and object-oriented methodologies.

Mark Toleman is an Associate Professor of Information Systems at the University of Southern Queensland, Australia where he has taught computing subjects to engineers, scientists, and business students for 18 years. He has a Bachelor of Applied Science (Distinction) and a Graduate Diploma in Information Processing from the Darling Downs Institute of Advanced Education, and a Master of Science from James Cook University. He also has a Ph.D. in computer science from the University of Queensland and has published more than 60 articles in books, refereed journals and refereed conference proceedings. His research interests include agile software development methodologies, component-based software development, e-business, the researcher-practitioner nexus, novice developers, philosophy of research, HCI, usability and software tool development and evaluation